Chapter 3 Convolutional Codes and Trellis Coded Modulation

• 3.1 Encoder Structure and Trellis Representation
• 3.2 Systematic Convolutional Codes
• 3.3 Viterbi Decoding Algorithm
• 3.4 BCJR Decoding Algorithm
• 3.5 Trellis Coded Modulation
§ 3.1 Encoder Structure and Trellis Representation

• Introduction
  – Encoder: contains memory (order \( m: m \) memory units);
  – Output: encoder output at time unit \( t \) depends on the input and the memory units status at time unit \( t \);
  – By increasing the memory order \( m \), one can increase the convolutional code’s minimum distance (\( d_{\text{min}} \)) and achieve low bit error rate performance (\( P_b \));
  – Decoding Methods:
    • Viterbi algorithm [1]: Maximum Likelihood (ML) decoding algorithm;
    • Bahl, Cocke, Jelinek, and Raviv (BCJR) [2] algorithm: Maximum \( A \) Posteriori Probability (MAP) decoding algorithm, used for iterative decoding process, e.g. turbo decoding.


§ 3.1 Encoder Structure and Trellis Representation

- The $(7, 5)_8$ conv. code
  - Encoder structure:
    - [Diagram of encoder structure]
  - Encoding Process:
    - (Initialised state $s_0s_1 = 00$)
    - At time $t_1$
      - [Diagram showing state transitions at $t_1$]
    - At time $t_2$
      - [Diagram showing state transitions at $t_2$]
    - At time $t_3$
      - [Diagram showing state transitions at $t_3$]

- Code rate: $\frac{1}{2}$;
- Memory: $m = 2$;
- Constraint length: $m + 1 = 3$
- Output calculation:
  - $c_1 = a \oplus S_0 \oplus S_1$;
  - $c_2 = a \oplus S_1$;
- Registers update:
  - $S_1' = S_0$.
  - $S_0' = a$. 

The state transitions at each time step are shown in the diagrams. The encoder structure and the encoding process are illustrated with the states $S_0$ and $S_1$, and the output calculations for $c_1$ and $c_2$ are provided. The registers are updated based on the encoded bits.
§ 3.1 Encoder Structure and Trellis Representation

At time $t_4$

At time $t_5$

At time $t_6$

Input sequence $[m_1 \ m_2 \ m_3 \ m_4 \ m_5 \ m_6] = [1 \ 0 \ 1 \ 0 \ 0 \ 0]$

Output sequence $[c_1^1 \ c_1^2 \ c_2^1 \ c_2^2 \ c_3^1 \ c_3^2 \ c_4^1 \ c_4^2 \ c_5^1 \ c_5^2 \ c_6^1 \ c_6^2] = [11 \ 10 \ 00 \ 10 \ 11 \ 00]$
§ 3.1 Encoder Structure and Trellis Representation

A state transition diagram of the $(7, 5)_8$ conv. code

State definition $(s_0 s_1)$
- $a = 00$
- $b = 10$
- $c = 01$
- $d = 11$

Interpretation of the state diagram
- Input bit (0) / output bits (11)
  - The current state of the encoder is $c$. If the input bit is 0, it will output 11 and the next state of the encoder is $a$. 

The current state of the encoder is $c$. If the input bit is 0, it will output 11 and the next state of the encoder is $a$. 
§ 3.1 Encoder Structure and Trellis Representation

Tree Representation of the \((7, 5)_8\) conv. code

Time unit: 1 2 3 4

Initialised state: a

Tree diagram interpretation:

The current state of the encoder is b. If the input bit is 0, the output will be 10, and the next state of the encoder is c.

Example 3.1 Determine the codeword that corresponds to message [0 1 1 0 1]
§ 3.1 Encoder Structure and Trellis Representation

Trellis of the (7, 5)₈ conv. code

<table>
<thead>
<tr>
<th>IN</th>
<th>Current State</th>
<th>Next State</th>
<th>Out</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>00</td>
<td>10</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>01</td>
<td>00</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>10</td>
<td>00</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>01</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>11</td>
<td>01</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>01</td>
<td>01</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>8</td>
</tr>
</tbody>
</table>

**Remark:** A trellis tells the state transition and IN/OUT relationship. It can be used to yield a convolutional codeword of a sequential input.

**Example 3.2** Use the above trellis to determine the codeword that corresponds to message [0 1 1 0 1].
§ 3.1 Encoder Structure and Trellis Representation

A number of conv. codes

(7, 5)_8 conv. code

(23, 35)_8 conv. code

(15, 13)_8 conv. code

(171, 133)_8 conv. code

4 states

16 states

8 states

64 states

Remark: A convolutional code’s error-correction capability improves by increasing the number of the encoder states.
§ 3.1 Encoder Structure and Trellis Representation

Remark: The encoder structure can also be represented by generator sequences or transfer functions.

Example 3.3: The $(7, 5)_8$ conv. code can also be written as:

A rate $1/2$ conv. code with generator sequences

\[ g^{(0)} = [1 \ 1 \ 1], \quad g^{(1)} = [1 \ 0 \ 1]. \]

A rate $1/2$ conv. code with transfer functions:

\[ g^{(1)}(x) = 1 + x + x^2, \quad g^{(2)}(x) = 1 + x^2. \]
§ 3.2 Systematic Convolutional Codes

- The $(7, 5)_8$ conv. code’s systematic counterpart is:

\[(7, 5)_8 \text{ conv. code} \quad S_0 \quad S_1 \quad c_1 \quad a \quad S_0' \quad S_1' \quad c_2\]

Non-systematic code

\[(1, 5/7)_8 \text{ conv. code} \quad a \quad (1, 5/7)_8 \text{ conv. code} \quad S_0 \quad S_1 \quad c_1 \quad c_2\]

Systematic code

Encoding and Registers’ updating rules:
[S_0, S_1] are initialization as [0 0];
\[c_1 = a; \quad \text{(systematic feature)} \quad \text{feedback} = S_0 \oplus S_1; \quad S_1' = S_0; \quad S_0' = a \oplus \text{feedback};\]

Remark: Systematic encoding structure is important for iterative decoding, e.g., the decoding of turbo codes.
§ 3.2 Systematic Convolutional Codes

For the $(1, 5/7)_8$ conv. code

<table>
<thead>
<tr>
<th>IN</th>
<th>Current State</th>
<th>Next State</th>
<th>Out</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>00</td>
<td>10</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>01</td>
<td>10</td>
<td>00</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>00</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>11</td>
<td>01</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>01</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
<td>01</td>
<td>01</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>8</td>
</tr>
</tbody>
</table>

Trellis

OUT

---

IN: 0

---

IN: 1
§ 3.3 Viterbi Decoding Algorithm

Let us extend the trellis of the $(7, 5)_8$ conv. code as if there is a sequential input.

- Such an extension results in a **Viterbi trellis**
- A path in the Viterbi trellis represents a convolutional codeword that corresponds to a sequential input (message).
§ 3.3 Viterbi Decoding Algorithm

- Decoding motivation: Given a received word $\bar{R}$, find the mostly likely codeword $\hat{C}$ such that the Hamming distance $d_{Ham}(\bar{R}, \hat{C})$ is minimized.
- Since $\hat{C}$ corresponds to a path in the Viterbi trellis, trace back the path of $\hat{C}$ enable us to find out the message.
- Branch metrics: Hamming distance between a transition branch’s output and the corresponding received symbol (or bits).
- Path metrics: Accumulated Hamming distance of the previous branch metrics.
§ 3.3 Viterbi Decoding Algorithm

**Example 3.4.** Given the (7, 5)₈ conv. code as in Examples 3.1-3.3. The transmitted codeword is
\[ \tilde{C} = [0\ 0\ 1\ 1\ 01\ 01\ 00]. \] After channel, the received word is
\[ \tilde{R} = [0\ 0\ 1\ 1\ [1]\ 01\ 00]. \] Try to use the Viterbi trellis to decode it.

Step 1: Calculate all the branch metrics.
§ 3.3 Viterbi Decoding Algorithm

Step 2: Calculate the path metrics.

When two paths join in a node, keep the smaller accumulated Hamming distance.

When the two joining paths give the same accumulated Hamming distance, pick up one randomly.
Step 3: Pick up the minimal path metric and trace back to determine the message.

Tracing rules: (1) Trellis connection;
   (2) The previous path metric should NOT be greater than the current path metric;
   (3) The tracing route should match the trellis transition ID.

Decoding output: 0 1 1 0 1
### § 3.3 Viterbi Decoding Algorithm

<table>
<thead>
<tr>
<th>Branch Metrics Table</th>
<th>Path Metrics Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 2 2 1 0</td>
<td>0 0 2 3 2 2</td>
</tr>
<tr>
<td>∞ ∞ 0 1 2</td>
<td>∞ ∞ 3 1 1 3</td>
</tr>
<tr>
<td>2 0 0 1 2</td>
<td>∞ 2 0 2 2 1</td>
</tr>
<tr>
<td>∞ ∞ 2 1 0</td>
<td>∞ ∞ 3 1 2 3</td>
</tr>
<tr>
<td>∞ 1 1 2 1</td>
<td></td>
</tr>
<tr>
<td>∞ 1 0 1</td>
<td></td>
</tr>
<tr>
<td>∞ 1 1 0 1</td>
<td></td>
</tr>
<tr>
<td>∞ ∞ 1 2 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trellis Transition ID Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 3 3 1</td>
</tr>
<tr>
<td>× 5 5 7 5</td>
</tr>
<tr>
<td>2 2 2 4 4</td>
</tr>
<tr>
<td>× 6 6 6 8</td>
</tr>
</tbody>
</table>
§ 3.3 Viterbi Decoding Algorithm

- **Soft-decision Viterbi decoding**
  - While we are performing the hard-decision Viterbi decoding, we have the scenario that two joining paths yield the same accumulated Hamming distance. This would cause decoding ‘ambiguity’ and performance penalty;
  - Such a performance loss can be compensated by utilizing soft-decision decoding, e.g., soft-decision Viterbi decoding

- **Modulation and Demodulation (e.g., BPSK)**
  - Modulation: mapping binary information into a transmitted symbol;
  - Demodulation: determining the binary information with a received symbol;
§ 3.3 Viterbi Decoding Algorithm

- Modulation and Demodulation (e.g., BPSK)

- Euclidean Distance

**Definition:** The Euclidean distance between points \( p \) and \( q \) is the length of the line segment connecting them.

\[
d_{Eud} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}
\]

**Hard-decision:** the information bit is 0. The Hamming distance becomes the Viterbi decoding metrics;

**Soft-decision:** the information bit has Pr. of 0.7 being 0 and Pr. of 0.3 bing 1. The *Euclidean distance (or probability)* becomes the Viterbi decoding metrics;

Received symbol

\[ (0.5, 0.9) \]

\[ (1, 0) \]

\[ (-1, 0) \]
Example 3.5. Given the \((7, 5)_{8}\) conv. code as in Examples 3.1-3.3. The transmitted codeword is \(\bar{C} = [0\ 0\ 1\ 1\ 0\ 1\ 0\ 0]\).

After BPSK modulation, the transmitted symbols are:
\((1, 0), (1, 0), (-1, 0), (-1, 0), (1, 0), (-1, 0), (1, 0), (1, 0), (1, 0)\).

After the channel, the received symbols are:
\((0.8, 0.2), (1.2, -0.4), (-1.3, 0.3), (-0.9, -0.1), (-0.5, 0.4), (-1.0, 0.1),
(1.1, 0.4), (-0.7, -0.2), (1.2, 0.2), (0.9, 0.3)\).
§ 3.3 Viterbi Decoding Algorithm

Step 1: Calculate all the branch metrics.

\[
\begin{array}{cccccc}
(0.8, 0.2) & (-1.3, 0.3) & (-0.5, 0.4) & (1.1, 0.4) & (1.2, 0.2) \\
(1.2, -0.4) & (-0.9, -0.1) & (-1.0, 0.1) & (-0.7, -0.2) & (0.9, 0.3)
\end{array}
\]
§ 3.3 Viterbi Decoding Algorithm

Step 2: Calculate the path metrics.

When two paths join in a node, keep the smaller accumulated Euclidean distance.
Step 3: Pick up the minimal path metric and trace back to determine the message.

Tracing rules: The same as hard-decision Viterbi decoding algorithm.

Decoding output: 0 1 1 0 1
§ 3.3 Viterbi Decoding Algorithm

<table>
<thead>
<tr>
<th>Branch Metrics Table</th>
<th>Path Metrics Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.53  3  2.53  1.76  0.42</td>
<td>0  0.53  3.53  5.48  5.25  5.67</td>
</tr>
<tr>
<td>∞  ∞  0.65  2.17  2.93</td>
<td>∞  ∞  4.83  3.08  3.64  6.65</td>
</tr>
<tr>
<td>2.88  0.45  0.65  2.17  2.93</td>
<td>∞  2.88  0.98  4.18  4.84  4.06</td>
</tr>
<tr>
<td>∞  ∞  2.53  1.76  0.42</td>
<td>∞  ∞  5.2   2.54  5.28  6.78</td>
</tr>
<tr>
<td>∞  ∞  1.95  2.10  2.74  2.68</td>
<td></td>
</tr>
<tr>
<td>∞  ∞  1.56  1.10  1.94</td>
<td></td>
</tr>
<tr>
<td>∞  ∞  2.32  1.56  1.10  1.94</td>
<td></td>
</tr>
<tr>
<td>∞  ∞  2.10  2.74  2.68</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trellis Transition ID Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  1  3  3  1</td>
</tr>
<tr>
<td>×  5  5  7  5</td>
</tr>
<tr>
<td>2  2  2  4  4</td>
</tr>
<tr>
<td>×  6  6  6  8</td>
</tr>
</tbody>
</table>
§ 3.3 Viterbi Decoding Algorithm

Free distance of convolutional code

- A convolutional code’s performance is determined by its free distance.
- Free distance

\[ d_{free} = \min\{d_{Ham}(\bar{C}_1, \bar{C}_2), \bar{C}_1 \neq \bar{C}_2\} \]

- With knowing \( \bar{C}' = [0 \ 0 \ 0 \ \ldots \ 0] \) is also a convolutional codeword.

\[ d_{free} = \min\{\text{weight}(\bar{C}), \bar{C} \neq 0\} \]

Hence, it is the minimum weight of all finite length paths in the Viterbi trellis that diverge from and emerge with the all zero state.
Hence, it is the minimum weight of all finite length paths in the Viterbi trellis that diverge from and emerge with the all zero state.

\[ d_{free} = 5 \]

**Remark:** Convolutional code with a large number of states will have a great \( d_{free} \), and hence stronger error-correction capability.
§ 3.3 Viterbi Decoding Algorithm

Remark: Convolutional code is more competent in correcting spread errors, but not bursty errors.

E.g., with \( \bar{R}_1 = [0 \ 1 \ e \ 1 \ \boxed{e} \ 1 \ 0 \ 1 \ 0 \ 0 \ e \ 1] \)

and \( \bar{R}_2 = [0 \ 1 \ 0 \ e \ e \ e \ 0 \ 1 \ 0 \ 0 \ 1 \ 1] \),

Viterbi algorithm is more competent in correcting received vector \( \bar{R}_1 \)
§ 3.4 BCJR Decoding Algorithm

  BCJR Algorithm: a Soft-In-Soft-Out (SISO) decoding.

- A Soft-In-Soft-Out (SISO) decoding algorithm that takes probabilities as the input and delivers probabilities as the output.

- With an attempt to deliver both the \textit{a posteriori} probabilities of $P(c_t | \bar{y})$ and $P(m_{t'} | \bar{y})$, it is also called the maximum \textit{a posteriori} (MAP) algorithm.

  - $c_t$ — convolutional coded bit, $t = 1, 2, \cdots n$.
  - $m_{t'}$ — information bit, $t' = 1, 2, \cdots k$.
  - $\bar{y}$ — received symbol vector.
§ 3.4 BCJR Decoding Algorithm

- In a trellis (e.g., trellis of the \((7, 5)_8\) conv. code).

The \((\text{IN, OUT, current state, next state})\) tuple happens as an entity.

That says at time instant \(t'\)

\[
\Sigma \ \text{Prob [trellis transition w.r.t. an input } \theta \text{ ]} = \text{Prob } [m_{t'} = \theta ], \ \theta \in \{0, 1\}.
\]

\[
\Sigma \ \text{Prob [trellis transition w.r.t. an output of } \theta \text{ ]} = \text{Prob } [c_{t'}^{(2)} = \theta ], \ \theta \in \{0, 1\}.
\]

We seek to determine all the \(\text{Prob [trellis transition w.r.t. an input of } \theta \text{ ] at time } t'\) to know

\[
P(m_{t'} = \theta | \bar{y}).
\]

We seek to determine all the \(\text{Prob [trellis transition w.r.t. an output of } \theta \text{ ] at time } t'\) to know

\[
P(c_{t'}^{(2)} = \theta | \bar{y}).
\]
§ 3.4 BCJR Decoding Algorithm

A Viterbi trellis snapshot at time instant $t'$:

- For a rate half conv. code, $m_{t'} \rightarrow c_{t'}^1, c_{t'}^2$
- Trellis state transition probability: $(\Omega, \Omega') \in \{a, b, c, d\}$

\[
\Gamma_{\Omega \rightarrow \Omega'} = P_a(m_{t'}) \cdot P_{ch}(c_{t'}^1) \cdot P_{ch}(c_{t'}^2)
\]
§ 3.4 BCJR Decoding Algorithm

- Determine the state transition probabilities.

\[ \Gamma_{\Omega \rightarrow \Omega'} = P_a(m_{t'}) \cdot P_{ch}(c_{t'}^1) \cdot P_{ch}(c_{t'}^2) \]

\[ P_{ch}(c_{t'}^1) = P(y_t | c_{t'}^1 = 0) = \frac{1}{\pi N_0} \exp\left(-\frac{||y_t - s_2||^2}{N_0}\right) \]

\[ P_{ch}(c_{t'}^1 = 1) = P(y_t | c_{t'}^1 = 1) = \frac{1}{\pi N_0} \exp\left(-\frac{||y_t - s_1||^2}{N_0}\right) \]

Channel observations:
E.g., BPSK is used for modulation.

A priori prob. of information bit. E.g., w/o knowledge of \( m_{t'} \),

\[ P_a(m_{t'} = 0) = P_a(m_{t'} = 1) = 0.5. \]
§ 3.4 BCJR Decoding Algorithm

- Determine the probability of each beginning state.

- Probability of beginning a trellis transition \((\Omega \to \Omega')\) from state \(\Omega\) (Determined by a forward trace).

\[
A_{t'}(\Omega) = N_A \sum_{(\Omega_0, \Omega)} A_{t' - 1}(\Omega_0) \cdot \Gamma_{\Omega_0 \to \Omega},
\]

\(t' = 1, 2, \ldots, k\).

\(N_A\): Normalization factor that ensures

\[A_{t'}(a) + A_{t'}(b) + A_{t'}(c) + A_{t'}(d) = 1.\]

- Knowing the Viterbi trellis starts from the all-zero state, we initialize:

\[A_0(a) = 1, \text{ and } A_0(b) = A_0(c) = A_0(d) = 0.\]

- E.g., in the highlighted trellis transition

\[A_{t'}(a) = A_{t' - 1}(a) \cdot \Gamma_{a \to a} + A_{t' - 1}(b) \cdot \Gamma_{b \to a}.\]
3.4 BCJR Decoding Algorithm

- Determine the probability of each ending state.

- Probability of ending the trellis transition ($\Omega \rightarrow \Omega'$) at state $\Omega'$ (Determined by a backward trace).

$$B_{t'+1}(\Omega') = N_B \sum_{(\Omega', \Omega'')} B_{t'+2}(\Omega'') \cdot \Gamma_{\Omega' \rightarrow \Omega''}.$$ 

$N_B$: Normalization factor that ensures $B_{t'+1}(a) + B_{t'+1}(b) + B_{t'+1}(c) + B_{t'+1}(d) = 1.$

- By ensuring after encoding, the shift registers (encoder) are restored to the all zero state (achieved by bit tailing), we can initialize:

$$B_{k+1}(a) = 1, \text{ and } B_{k+1}(b) = B_{k+1}(c) = B_{k+1}(d) = 0.$$ 

- E.g., in the highlighted trellis transition

$$B_{t'+1}(c) = B_{t'+2}(b) \cdot \Gamma_{c \rightarrow b} + B_{t'+2}(d) \cdot \Gamma_{c \rightarrow d}.$$ 

IN: 0

IN: 1

---

---

IN: 0

IN: 1
§ 3.4 BCJR Decoding Algorithm

- Determine the a posteriori probability of each information bit

- After the Forward Trace and Backward Trace, we obtain all the $A_t'(\Omega)$, $B_{t'+1}(\Omega')$ and $\Gamma_{\Omega \rightarrow \Omega'}$ of each time instant $t'$. We can now determine the a posteriori probabilities $P(m_{t'}|\overline{y})$ for each information bit as

$$P(m_{t'} = 0|\overline{y}) = N_p \sum_{(\Omega, \Omega')} A_t'(\Omega) \cdot \Gamma_{\Omega \rightarrow \Omega'} \cdot B_{t'+1}(\Omega')$$

$$P(m_{t'} = 1|\overline{y}) = N_p \sum_{(\Omega, \Omega')} A_t'(\Omega) \cdot \Gamma_{\Omega \rightarrow \Omega'} \cdot B_{t'+1}(\Omega')$$

$N_p$: Normalization factor that ensures $P(m_{t'} = 0|\overline{y}) + P(m_{t'} = 1|\overline{y}) = 1$. 
3.4 BCJR Decoding Algorithm

- E.g.,

\[ P(m_{t^{'}} = 0|\bar{y}) = N_P \cdot (A_{t^{'}}(a) \cdot \Gamma_{a \rightarrow a} \cdot B_{t^{'}}(a) + A_{t^{'}}(b) \cdot \Gamma_{b \rightarrow a} \cdot B_{t^{'}}(a)) \]

\[ A_{t^{'}}(c) \cdot \Gamma_{c \rightarrow b} \cdot B_{t^{'}}(b) + A_{t^{'}}(d) \cdot \Gamma_{d \rightarrow b} \cdot B_{t^{'}}(b)) \].

\[ N_P = P(m_{t^{'}} = 0|\bar{y}) + P(m_{t^{'}} = 1|\bar{y}) \].

- Decision based on the \textit{a posteriori} probabilities.

\[ \hat{m}_{t^{'}} = 0, \text{ if } P(m_{t^{'}} = 0|\bar{y}) \geq P(m_{t^{'}} = 1|\bar{y}) \]

\[ \hat{m}_{t^{'}} = 1, \text{ if } P(m_{t^{'}} = 1|\bar{y}) > P(m_{t^{'}} = 0|\bar{y}) \].
Example 3.6. With the same transmitted codeword and received symbols of Example 3.5, use the BCJR algorithm to decode it.

With the received symbols, we can determine

\[
\begin{align*}
P_{ch}(c_1^1 = 0) &= 0.83 & P_{ch}(c_1^2 = 0) &= 0.92 & P_{ch}(c_2^1 = 0) &= 0.07 & P_{ch}(c_2^2 = 0) &= 0.14 \\
P_{ch}(c_1^1 = 1) &= 0.17 & P_{ch}(c_1^2 = 1) &= 0.08 & P_{ch}(c_2^1 = 1) &= 0.93 & P_{ch}(c_2^2 = 1) &= 0.86 \\
P_{ch}(c_3^1 = 0) &= 0.27 & P_{ch}(c_3^2 = 0) &= 0.12 & P_{ch}(c_4^1 = 0) &= 0.90 & P_{ch}(c_4^2 = 0) &= 0.20 \\
P_{ch}(c_3^1 = 1) &= 0.73 & P_{ch}(c_3^2 = 1) &= 0.88 & P_{ch}(c_4^1 = 1) &= 0.10 & P_{ch}(c_4^2 = 1) &= 0.80 \\
P_{ch}(c_5^1 = 0) &= 0.92 & P_{ch}(c_5^2 = 0) &= 0.86 \\
P_{ch}(c_5^1 = 1) &= 0.08 & P_{ch}(c_5^2 = 1) &= 0.14
\end{align*}
\]
§ 3.4 BCJR Decoding Algorithm

Step 1: Determine $\Gamma_{\Omega \rightarrow \Omega'}$ of all transitions.
§ 3.4 BCJR Decoding Algorithm

Step 2: Forward trace, determine $A_{t'}(\Omega)$ of values.
§ 3.4 BCJR Decoding Algorithm

Step 3: Backward Trace, determine $B_{t'+1}(\Omega')$ of values.

Assume we know the trellis ends at state $c$. 
3.4 BCJR Decoding Algorithm

Step 4: Determine the \textit{a posteriori} probabilities of each information bit.

\[
\begin{align*}
P(m_1 = 0 | \bar{y}) &= 1.00 & \Rightarrow & \hat{m}_1 = 0 \\
P(m_1 = 1 | \bar{y}) &= 0.00 \\
\end{align*}
\]

\[
\begin{align*}
P(m_2 = 0 | \bar{y}) &= 0.00 & \Rightarrow & \hat{m}_2 = 1 \\
P(m_2 = 1 | \bar{y}) &= 1.00 \\
\end{align*}
\]

\[
\begin{align*}
P(m_3 = 0 | \bar{y}) &= 0.00 & \Rightarrow & \hat{m}_3 = 1 \\
P(m_3 = 1 | \bar{y}) &= 1.00 \\
\end{align*}
\]

\[
\begin{align*}
P(m_4 = 0 | \bar{y}) &= 1.00 & \Rightarrow & \hat{m}_4 = 0 \\
P(m_4 = 1 | \bar{y}) &= 0.00 \\
\end{align*}
\]

\[
\begin{align*}
P(m_5 = 0 | \bar{y}) &= 0.00 & \Rightarrow & \hat{m}_5 = 1 \\
P(m_5 = 1 | \bar{y}) &= 1.00 \\
\end{align*}
\]
§ 3.4 BCJR Decoding Algorithm

BER performance of $(7, 5)_8$ conv. code over AWGN channel using BPSK.
BER performance of different conv. code over AWGN channel using BPSK.
3.5 Trellis Coded Modulation

- Convolutional code enables reliable communications. But as a channel code, its error-correction function is on the expense of spectral efficiency.

- Spectral efficiency ($\eta$) = \( \frac{\text{Nr. of information bits}}{\text{transmitted symbol}} \)

- E.g., an uncoded system using BPSK
- A rate 1/2 conv. coded system using BPSK

  \( \eta = 1 \) info bits/symbol
  \( \eta = 0.5 \) info bits/symbol

- Can we achieve reliable and yet spectrally efficient communication?

  **Solution:** Trellis Coded Modulation (TCM) that integrates a conv. code with a high order modulation [3].

---

§ 3.5 Trellis Coded Modulation

- A general structure of the TCM scheme

\[ \text{Rate } \frac{k}{k+1} \]
\[ \text{conv. encoder} \]

Select a subset from the constellation

Select a point from the subset

Output symbol
§ 3.5 Trellis Coded Modulation

- A rate 2/3 TCM code.

Rate ½ 4-state Convolutional Code

8PSK Constellation
### § 3.5 Trellis Coded Modulation

#### State table of the rate 2/3 TCM code

<table>
<thead>
<tr>
<th>Input</th>
<th>Current State</th>
<th>Next State</th>
<th>Output</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$, $a_2$</td>
<td>$S_1$, $S_2$</td>
<td>$S_1'$, $S_2'$</td>
<td>$c_1$, $c_2$, $c_3$</td>
<td>8PSK sym</td>
</tr>
<tr>
<td>0, 0</td>
<td>0, 0</td>
<td>0, 0</td>
<td>0, 0</td>
<td>0</td>
</tr>
<tr>
<td>1, 0</td>
<td>0, 0</td>
<td>0, 1</td>
<td>0, 0</td>
<td>2</td>
</tr>
<tr>
<td>0, 1</td>
<td>0, 0</td>
<td>0, 0</td>
<td>0, 0</td>
<td>4</td>
</tr>
<tr>
<td>1, 1</td>
<td>0, 0</td>
<td>0, 1</td>
<td>0, 1</td>
<td>6</td>
</tr>
<tr>
<td>0, 0</td>
<td>0, 1</td>
<td>1, 1</td>
<td>0, 1</td>
<td>1</td>
</tr>
<tr>
<td>1, 0</td>
<td>0, 1</td>
<td>1, 1</td>
<td>0, 1</td>
<td>3</td>
</tr>
<tr>
<td>0, 1</td>
<td>1, 0</td>
<td>0, 1</td>
<td>0, 1</td>
<td>5</td>
</tr>
<tr>
<td>1, 1</td>
<td>1, 0</td>
<td>1, 1</td>
<td>1, 1</td>
<td>7</td>
</tr>
<tr>
<td>0, 0</td>
<td>0, 1</td>
<td>0, 1</td>
<td>0, 0</td>
<td>0</td>
</tr>
<tr>
<td>1, 0</td>
<td>1, 0</td>
<td>0, 1</td>
<td>0, 1</td>
<td>2</td>
</tr>
<tr>
<td>0, 1</td>
<td>1, 0</td>
<td>0, 1</td>
<td>0, 1</td>
<td>4</td>
</tr>
<tr>
<td>1, 1</td>
<td>1, 0</td>
<td>0, 1</td>
<td>1, 1</td>
<td>6</td>
</tr>
<tr>
<td>0, 0</td>
<td>1, 1</td>
<td>1, 1</td>
<td>0, 0</td>
<td>1</td>
</tr>
<tr>
<td>1, 0</td>
<td>1, 1</td>
<td>1, 1</td>
<td>0, 0</td>
<td>3</td>
</tr>
<tr>
<td>0, 1</td>
<td>1, 1</td>
<td>1, 1</td>
<td>1, 0</td>
<td>5</td>
</tr>
<tr>
<td>1, 1</td>
<td>1, 1</td>
<td>1, 0</td>
<td>1, 1</td>
<td>7</td>
</tr>
</tbody>
</table>
§ 3.5 Trellis Coded Modulation

- Set Partitioning 8PSK

$c_3c_2c_1 = 010 \quad c_1 = 0$

$c_3c_2c_1 = 001 \quad c_1 = 1$

$\Delta_0 = 2\varepsilon' \sin \frac{\pi}{8} = 0.765\varepsilon'$

$\Delta_1 = 2\varepsilon' \sin \frac{\pi}{4} = \sqrt{2\varepsilon'} = 1.41\sqrt{\varepsilon'}$

Original constellation

Subset 1

Subset 2

(0, 4)  \quad 110 (2, 6)  \quad (1, 5)  \quad (3, 7)
§ 3.5 Trellis Coded Modulation

- Set Partitioning 8PSK

By doing set partitioning, the minimum distance between point within a subset is increasing as: \( \Delta_0 < \Delta_1 < \Delta_2 \).

\[
\Delta_0 = d(0,1) = 2\sqrt{\varepsilon'} \sin \frac{\pi}{8} = 0.765 \sqrt{\varepsilon'}
\]

\[
\Delta_1 = d(0,2) = \sqrt{2\varepsilon'} = 1.414 \sqrt{\varepsilon'}
\]

\[
\Delta_2 = d(0,4) = 2\sqrt{\varepsilon'}
\]
§ 3.5 Trellis Coded Modulation

- Viterbi trellis of the rate 2/3 TCM code

For diverse/remerge transition:

\[ d_{\text{free}}^2 = [d^2(0,2) + d^2(0,1) + d^2(0,2)] \]
\[ = 2\varepsilon' + (2 - \sqrt{2})\varepsilon' + 2\varepsilon' = 4.586\varepsilon' \]

For parallel transition:

\[ d_{\text{free}}^2 = d^2(0,4) = 4\varepsilon' \]

Choose the smaller one as the free distance of the code:

\[ d_{\text{free}}^2 = 4\varepsilon' \]

**Remark:** Bit \( c_3 = 0 \) and \( c_3 = 1 \) result in two parallel transition branches. By doing set partitioning, we are trying to maximize the Euclidean distance between the two parallel branches. So that the free distance of the TCM code can be maximized.
3.5 Trellis Coded Modulation

- Asymptotic coding gain over an uncoded system.
- Spectral efficiency ($\eta$) = 2 info bits/sym.

Asymptotic coding gain over an uncoded system.

Remark: With the same transmission spectral efficiency of 2 info bits/sym, the TCM coded system achieves 3 dB coding gain over the uncoded system asymptotically.